VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Bakalářská práce

2012 Michal Pavlíček

VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2012 Michal Pavlíček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Pavlíček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

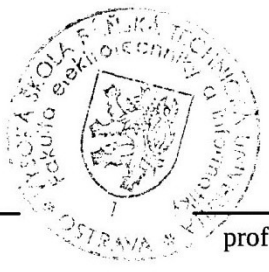
Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radim Bača, Ph.D.**

Konzultant bakalářské práce: Bc. Ondřej Kvasnovský

Datum zadání: 19.11.2010

Datum odevzdání: 04.05.2012



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

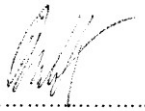
prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 4. května 2012



.....

Podpis

Poděkování

Na tomto místě bych rád poděkoval panu Bc. Ondřeji Kvasnovskému za příležitost vykonání odborné praxe ve firmě Tieto Czech s.r.o., svým kolegům, kteří se podíleli na vývoji se mnou, Ing. Peteru Almásymu za trpělivost a všem zaměstnancům firmy Tieto Czech s.r.o., za jejich ochotu a odbornou pomoc.

Abstrakt

Tato bakalářská práce se věnuje průběhu odborné praxe ve firmě Tieto Czech s.r.o. Pracoval jsem na pozici softwarového vývojáře v jazyce Java a po celou dobu trvání praxe jsem pracoval na informačním systému „My Wires“. Vývoj systému začal na zelené louce a bez jasné představy o jeho funkcionalitě. Hlavní myšlenkou bylo vyvinout systém pro evidenci a správu fyzického připojení počítačů do podnikové ethernetové sítě pro potřeby správců počítačových sítí. Tato práce se stručně věnuje použitým technologiím, popisu informačního systému a samotnému vývoji. Na závěr jsou shrnuty přínosy odborné praxe.

Klíčová slova

Tieto, Java EE, Vaadin, Spring, Hibernate, Maven, informační system, praxe

Abstract

The Bachelor´s Thesis describes the practical experience I received in Tieto Czech s.r.o. I was working there as a Java software developer. I was working on the „My Wires“ information system all my stay in the company. The System has been developed from scratch and without any clear idea of its functionality. The main idea was to develop a system for network administrators which would be used to record and manage physical connections of computers to corporate Ethernet network. The thesis briefly describes the system, technologies used and its actual development. In conclusion, benefits of professional practice are summarized.

Key words

Tieto, Java EE, Vaadin, Spring, Hibernate, Maven, information system, practise

Seznam použitých symbolů a zkratek

AJAX – Asynchronous JavaScript and XML
AOP – Aspect Oriented Programming
API – Application Programming Interface
AWT – Abstract Window Toolkit
CRUD – Create, Read, Update, Delete
CSS – Cascading Style Sheet
CVS – Concurrent Version System
DAIS – Databázové a Informační Systémy
DAO – Data Access Object
DI – Dependency Injection
DML – Data Manipulation Language
EJB – Enterprise Java Bean
GWT – Google Web Toolkit
HSQLDB – HyperSQL Database
HTML – HyperText Markup Language
HQL – Hibernate Query Language
IDE – Integrated Development Environment
IoC – Inversion of Control
JAR – Java Archive
JAT – Java Technologie
Java EE – Java Enterprise Edition
JAXB – Java Architecture for XML Binding
JDBC – Java Database Connectivity
JOS – Java Open Source
JPA – Java Persistence API
JRE – Java Runtime Environment
JSF – Java Server Face
JSP – Java Server Page
JTA – Java Transaction API
JVM – Java Virtual Machine
LDAP – Lightweight Directory Access Protocol
MOJO – Maven Old Java Object
MVC – Model-View-Controller
OOP – Object Oriented Programming
ORM – Object Relational Mapping
OXM – Object/XML Mapping

PJI	– Programovací Jazyky I
POJO	– Plain Old Java Object
POM	– Project Object Model
RPC	– Remote Procedure Calling
SOAP	– Simple Object Access Protocol
SQL	– Structured Query Language
SVN	– Subversion
SWI	– Softwarové Inženýrství
SWT	– Standard Widget Toolkit
TCP/IP	– Transmission Control Protocol / Internet Protocol
TZD	– Teorie Zpracování Dat
UML	– Unified Modeling Language
UI	– User Interface
VIA	– Vývoj Internetových Aplikací
VIS	– Vývoj Informačních Systémů
VO	– View Object
WS	– Web Service
WSDL	– Web Services Description Language
XML	– Extensible Markup Language

Obsah

1 Úvod

Jako jeden z mála studentů jsem si jako alternativu bakalářské práce vybral možnost absolvování individuální odborné praxe ve společnosti Tieto Czech s.r.o. Pro tuto možnost jsem se rozhodl z důvodu absolvování odborného kurzu Tieto IT Academy – Java & Software development, kde jsem měl možnost nahlédnout pod pokličku vývoje software v praxi. Protože mě obsah a forma kurzu nadchla, neváhal jsem a přihlásil se i na odbornou praxi v této společnosti. Po zaslání životopisu jsem byl kontaktován a pozván na přijímací pohovor, kde mi účast v Tieto IT Academy dopomohla k přijetí.

Tato bakalářská práce se věnuje popisu informačního systému „My Wires“ vyvíjeného v rámci odborné praxe, jeho architektuře a návrhu a také budou zmíněny technologie, které byly při implementaci použity. V první části jsou zmíněny právě technologie, kde každé věnuji pár řádku pro jejich přiblížení, protože částečné pochopení těchto technologií bylo nutnou podmínkou pro aktivní podíl na vývoji daného systému. Poté je krátce popsána architektura, dále popis jednotlivých balíčků a na závěr je popsán samotný informační systém ve své finální podobě.

2 Odborné zaměření firmy

2.1 Tieto Czech s.r.o.

Společnost Tieto Czech s.r.o. je přední severoevropská společnost poskytující kompletní služby v oblasti inovace, návrhu, vývoje, implementace, systémové integrace, údržby a outsourcingu IS/IT a produktového inženýrství. Dnes se svými téměř 18 000 zaměstnanci ve 22 zemích světa se stává přední společností nabízející integrace služeb. [1]

Historie společnosti sahá až do roku 1968, kde byla založena ve finském Espoo společnost Tietotehdas Oy zabývající se vývojem a údržbou IT systémů především pro finskou Union Bank a také pro lesní průmysl. V 90. letech zaznamenala rychlý rozvoj díky akvizicím, fúzím a vstupu do strategických aliancí. V roce 1995 změnila své jméno na TT Tieto a v roce 1998 na Tieto. Od roku 1999, kdy se spojila společnost Tieto se švédskou společností Enator, nesla jméno TietoEnator. Od roku 2009 nese společnost název Tieto Corporation. [2]

V České Republice má společnost Tieto Czech s.r.o. hlavní sídlo v Ostravě, kde zaměstnává téměř 2 000 zaměstnanců. Další pobočky v České republice jsou v Českých Budějovicích, Brně a také v Praze.

2.2 Pracovní zařazení ve firmě

Ve společnosti jsem vykonával pozici Software Developera v oddělení JOS (Java Open Source). Byl jsem se zbylýma dvěma studenty vykonávající odbornou praxi přiřazen na nově vznikající projekt nazvaný „My Wires“. Byl jsem součástí zprvu početného, přibližně 12-ti členného týmu, který se v polovině doby vykonávání odborné praxe zúžil na šest členů, tři studenty a tři zaměstnance. V únoru jsme již zůstali na vývoj systému jen sami - tři studenti vykonávající odbornou praxi. Hlavním úkolem bylo vytvořit funkční a odladěný základ modulárního, dobře škálovatelného informačního systému pro evidenci stavu a správu zásuvek RJ-45.

Na vývoji jsem se aktivně podílel specifikací požadavků, analýzou, implementací i testováním. Na závěr jsem se podílel i na dokumentaci daného informačního systému.

3 Úkoly zadané během odborné praxe

Po nástupu do společnosti Tieto Czech s.r.o. jsem byl seznámen s pracovištěm a budoucími kolegy, kteří rovněž vykonávali odbornou praxi. Dostali jsme každý osobní počítač včetně vybavení a byli jsme rozděleni do jednotlivých kanceláří, kde byla ještě volná místa k sezení. Po seznámení s pracovištěm, firemní strukturou a sítí jsme byli obeznámeni s požadavky a náplní naší odborné praxe.

Prvním úkolem pro každého studenta bylo nastudování určité technologie a následná prezentace nabytých znalostí zbývajícím studentům. Tento postup byl zvolen z toho důvodu, že představa o naším nastávajícím projektu nebyla z cela jasná, ale představa o použitých technologiích byla celkem zřejmá. Mým úkolem bylo nastudovat rámec pro objektově-relační mapování Hibernate, způsoby mapování, integraci s aplikačním rámcem Spring a nabyté znalosti demonstrovat na dvou typech entit s vazbou M:N. Zbylí dva kolegové měli za úkol výše zmíněný aplikační rámec Spring a rámec Vaadin, který je určen pro tvorbu Rich Internet Application (RIA). Jako společný úkol bylo použití vývojového prostředí Eclipse a dodržování struktury projektu podle definice Apache Maven.

Nicméně neuběhly ani dva týdny naší odborné praxe a situace se změnila. Tým se rozrostl o dalších devět členů, pravděpodobně nových zaměstnanců a byla nám všem představena myšlenka informačního systému pro evidenci a správu ethernetových zásuvek RJ-45, pro potřeby síťových administrátorů.

3.1 Přiřazení k projektu My Wires

Po přiřazení k týmu nám stále zůstaly úkoly zadané na začátku naší odborné praxe, nicméně přibyly další. Vzhledem k již početnému týmu se začalo se specifikací systému a následnou analýzou těchto požadavků. V tuto dobu jsem si vyzkoušel roli analytika a jsem upřímně rád, že v týmu byli i zkušenější kolegové, kteří mnoho problému vyšlých z analýzy vyřešili a já se mohl přiučit. Navrhla se doména, definovaly se práva a následně role, udělaly se use case diagramy a navrhla se architektura, která již stejně byla daná požadavkem na použité technologie. Po těchto úkonech se začalo s implementací, která byla doprovázena velice častým tzv. code review, kde s námi projektový manažer, Ondřej Kvasnovský, probíral zásady pro tvorbu čistého, samodokumentujícího se kódu.

Vzhledem k povaze a náplni odborné praxe je nemožné sepsat konkrétní seznam úkolů. Mojí náplní odborné praxe bylo naučit se pracovat s určitými technologiemi, podílet se na vývoji, jehož náplní byly někdy jednoduché úkoly, jako napsání jednotkových testů, implementace některých DAO tříd, mapperů či inicializátorů až po úkoly celkem obtížné, které zahrnovaly opravy chyb objevených až v pokročilém stádiu vývoje, či implementace jedné záložky pro administraci týmu.

4 Technologie

Protože částečná znalost níže uvedených technologií byla téměř nutnou součástí naší praxe, rozhodl jsem se je v této práci alespoň částečně přiblížit.

4.1 Apache Maven

[3] Maven je mocný nástroj pro správu, řízení a automatizaci sestavování (build) aplikací psaných v jazyce Java, nicméně nechybí podpora i pro jiné programovací jazyky jako je například jazyk C# či Ruby. Na rozdíl od nástroje Ant, který je určen ke stejným účelům, [4] Maven používá deklarativní definici projektu, kdežto Ant používá systém pravidel a jejich závislostí, takže základní rozdíl je tedy v tom, že Antu definujeme jakým způsobem má projekt sestavit, Mavenu čeho se má dosáhnout.

Maven popisuje strukturu projektu v souboru nazvaném *pom.xml*, který je umístěn v kořenovém adresáři daného projektu. Zkratka POM vychází z anglického Project Object Model. V *pom.xml* uvádíme všechny závislosti, plug-iny, reportování, systém pro správu verzí jako je například SVN nebo Git, průběžnou integraci – tzv. Continuous integration, bug tracing (např. Jira), dále můžeme doplnit informace o projektu či vývojářích podílejících se na projektu.

Veškeré balíčky se stahují z globálního repositáře a ukládají do lokální cache, protože sestavované projekty používají JAR (Java Archive) jen z lokální cache.

Mezi hlavní výhody tedy patří již zmiňovaná správa závislostí, snadné generování reportů jako je třeba kontrola pokrytí testy či vygenerování struktury projektu, podpora verzovacích systémů, nezávislost na IDE a také podpora webového serveru Jetty, díky kterému je možno nasadit webovou aplikaci přímo z příkazové řádky bez instalace servlet kontejneru.

4.2 Spring

Rámec Spring je modulární Java/JEE aplikační rámec, jenž také bývá někdy označován jako odlehčený kontejner. Využíván je stejně jako Enterprise JavaBeans (EJB) zejména pro tvorbu webových aplikací, ale lze jej použít v podstatě pro jakýkoliv typ aplikace, včetně klasických (desktop GUI) aplikací. Dalším důležitým cílem použití rámce Spring je snadná testovatelnost výsledné aplikace. Spring umožňuje čistým a pohodlným způsobem vzájemně oddělit (z hlediska vzájemné závislosti) nejen jednotlivé vrstvy, ale dokonce i jednotlivé objekty, což je klíčovou podmínkou pro možnost využití klasického jednotkového testování. Vzhledem k tomu, že agilní metodiky vývoje software, jsou stále populárnější a rozšířenější, je rovněž tato vlastnost rámce považována za klíčovou. Dále rámec konzistentním způsobem podporuje všechny vrstvy aplikací, prezentační vrstvou počínaje, přes aplikační až k datové a perzistentní vrstvě [5].

Objekty, které tvoří aplikaci, jsou během svého životního cyklu spravovány kontejnerem rámce Spring a výjimku netvoří ani objekty aplikační vrstvy. Spring spravuje aplikaci již na úrovni objektů, nikoliv velkých komponent, jako je tomu u EJB.

Ke službám, které může návrhář aplikace pro správu objektů využít, patří zejména jednotný procedurální i deklarativní transakční management, jednotný způsob konfigurace aplikace v době nasazení, pokročilá procedurální i deklarativní správa zabezpečení, správa provázání a závislostí objektů, pooling objektů a další. Stěžejními návrhovými technikami, které poskytování těchto služeb umožňují, jsou programování orientované na aspekty (AOP - Aspect Oriented Programming) a návrhový vzor Obrácení řízení (IoC - Inversion of Control). Implementace IoC, respektive návrhového vzoru Injektáž závislostí (DI - Dependency Injection) v rámci Spring představuje nejkomplexnější řešení této problémové oblasti vůbec.

Klíčovým pojmem celé architektury aplikačního rámce Spring je tzv. továrna tříd (bean factory). Továrna tříd má na starosti základní funkcionalitu rámce Spring, tedy zejména provázání objektů pomocí DI a transparentní aplikaci služeb rámce (transakce atd.) pomocí AOP. Továrna tříd po svém startu drží objekty spravované aplikace, vzájemně je prováže a poskytuje odkazy na tyto objekty svým klientům. To, které objekty jsou továrnou tříd spravovány, je definováno většinou prostřednictvím konfiguračního souboru ve formátu XML. Lze použít i jiné formáty, například tzv. properties soubor, ale zdaleka nejrozšířenější jsou XML soubory, které jsme použili i na našem projektu.

Spring se skládá z modulů, které jsou rozděleny do šesti hlavních skupin [6]:

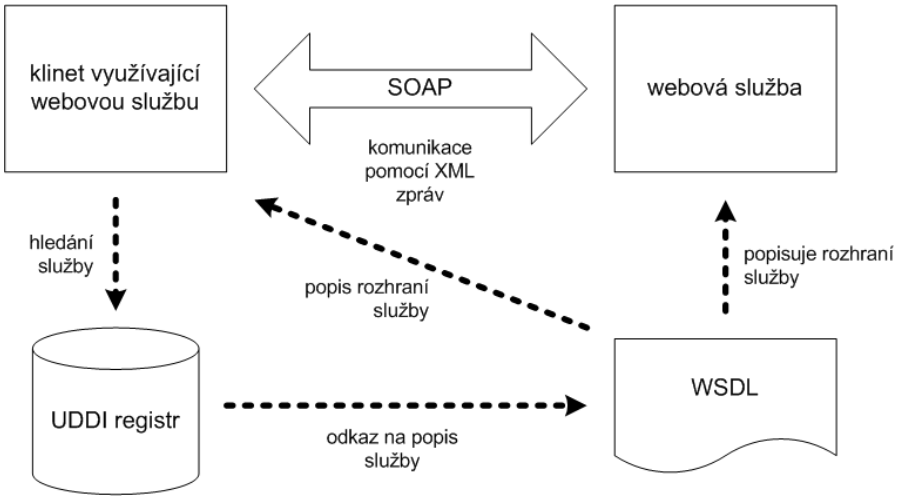
- Core container – obsahuje moduly Core, Beans, Context a Expression Language;
- Data Access / Integration – zde jsou moduly JDBC, ORM, OXM, JMS a Transactions;
- Web – skládá se z Web-Servlet, Web-Struts a Web-Portlet;
- AOP a Instrumentation – je modul implementující podporu pro aspektově orientované programování, umožňuje separovat částí kódu prolínající se celou aplikací (autorizace, logování, transakce) do takzvaných aspektů a jejich následnou aplikaci na jakýkoli POJO objekt. Využití AOP modulu se prolíná celým rámcem a jedná se o jednu z nejsilnějších vlastností Springu;
- Test – tento modul podporuje testování komponent Springu pomocí JUnit nebo TestNG.

4.3 Webové služby

Webové služby umožňují jednoduchou komunikaci mezi aplikacemi ve velmi heterogenním prostředí, protože komunikace je založena na platformě nezávislých standardech – především na jazyce XML a protokolu HTTP [7]. Protokol HTTP pro přenos SOAP (Simple Object Access Protocol) zprávy je použit z důvodu, že firewally již zavedených systémů nemají dostatek důvěry v SOAP protokol. Aplikace si mezi sebou posílají XML zprávy, které přenášejí dotazy a odpovědi jednotlivých aplikací. Celá infrastruktura webových služeb je založena na třech základních technologiích:

- SOAP – protokol používaný pro komunikaci;
- WSDL (Web Services Description Language) – standardní formát pro popis rozhraní webové služby;
- UDDI (Universal Description, Discovery and Integration) – standardní mechanismus umožňující registraci a vyhledávání webových služeb.

Vzájemné vztahy mezi těmito třemi technologiemi jsou zachycené na následujícím obrázku č. 1:



Obrázek 1 - Vztah základních technologií webových služeb [7]

Ke každé webové službě by měl být k dispozici její formální popis v jazyce WSDL. Z tohoto popisu již jde automaticky vygenerovat SOAP požadavek. Ve větších systémech nebo přímo v otevřeném prostředí Internetu se popis služby může zaregistrovat do UDDI registru. Ten slouží jako jakýsi telefonní seznam („zlaté stránky“), který umožňuje vyhledávání služeb s určitými parametry. Klient, který chce využít webovou službu, získá buď přes UDDI, nebo přímo její popis. Z něj je jasné, jakou strukturu má mít SOAP zpráva a kam se má webové službě poslat, aby ji rozpoznala [7].

SOAP je protokol pro posílání zpráv XML a je základem webových služeb. Ostatní standardy jako WSDL a UDDI vznikly až později po uvedení SOAPu a jen dále rozšiřují jeho možnosti a snadnost použití. SOAP umožňuje zaslání XML zprávy mezi dvěma aplikacemi a pracuje tedy na principu peer-to-peer. Zpráva je jednosměrný přenos informace od odesílatele k příjemci, ale díky kombinování několika zpráv můžeme pomocí SOAPu snadno implementovat běžné komunikační scénáře.

Nejčastěji se SOAP používá jako náhrada vzdáleného volání procedur (RPC), tedy v modelu požadavek/odpověď. Jedna aplikace pošle v XML zprávě požadavek druhé aplikaci, tak požadavek obslouží a výsledek zašle jako druhou zprávu zpět původnímu iniciátorovi komunikace. V tomto případě bývá webová služba vyvolána webovým serverem, který čeká na požadavky klientů a v okamžiku, kdy přes HTTP přijde SOAP zpráva, spustí webovou službu a předá jí požadavek. Výsledek služby je pak předán zpět klientovi jako odpověď [7].

4.4 Hibernate

Hibernate je nástroj poskytující techniku pro objektově-relační mapování tříd jazyka Java na tabulky relační databáze a naopak. Krom toho, Hibernate poskytuje nástroje, jak s těmito daty manipulovat včetně objektově orientovaného jazyka (HQL), který na základě objektové notace dokáže uložená data z databáze vybírat v podobě objektů. Díky tomu dokáže usnadnit práci s daty nad databází a snížit čas potřebný pro naprogramování podobných metod. Hibernate dále poskytuje mechanismy pro transakční zpracování dat, polymorfní perzistenci, kešování, objektově orientované sestavování dotazů, vlastní datové typy a mnoho další funkcionality. Hibernate poskytuje jednotné API pro práci s řadou databází a databázových serverů [5].

Hlavní pracovní jednotkou Hibernate je sezení. Jedná se o implementaci rozhraní *Session*. Sezení zapouzdřuje JDBC spojení a představuje konverzi mezi objektovou a relační reprezentací dat. Sezení dále slouží jako továrna pro transakce a udržuje cache první úrovně. Objekty typu sezení je levné vytvořit a uvolnit, protože tohle se děje velmi často, zpravidla minimálně jednou

během HTTP požadavku na webovou aplikaci. Rozhraní *Session* a *Transaction* jsou využívána aplikací pro zajištění základních operací nad databázovými tabulkami. Mezi základní operace řadíme příkazy DML jako je *INSERT*, *SELECT*, *UPDATE* a *DELETE*. Sezení je vytvářeno pomocí továrny sezení (rozhraní *SessionFactory*). Tato továrna je konfigurována pomocí nastavení uvedených v *hibernate.properties* a obsahuje zkompilované XML mapování tříd na tabulky relační databáze (pro konfiguraci mapování můžeme po vzoru JPA použít i objektové notace - anotace). Továrna sezení udržuje cache druhé úrovně. Objekt implementující toto rozhraní je poměrně drahé vytvořit, proto se v aplikaci většinou vyskytuje pouze jedna instance tohoto objektu. Pokud aplikace pracuje nad více databázemi, pro každou z nich musí existovat jedna továrna sezení. JDBC spojení pro danou továrnu sezení připravuje poskytovatel spojení. Jedná se o množinu spojení (pool of connections), kterou poskytovatel spojení udržuje a tato jednotlivá spojení následně poskytuje továrně sezení [5].

Rozhraní *Transaction* je doporučené API, ale není povinné. Toto transakční API odstiňuje aplikační kód od implementace transakcí na nižší úrovni. Jedná se např. o JDBC transakce nebo JTA. Toto odstínění napomáhá přenositelnosti aplikace. Transakce vytváří transakční továrna. Transakční továrnou rozumíme implementaci rozhraní *TransactionFactory*.

Na perzistentní třídy Hibernate neklade, na rozdíl od jiných ORM rámců, žádné speciální nároky. Perzistentní třídy jsou obyčejné objekty, které musí obsahovat bezparametrický konstruktor. Je doporučováno, aby tyto třídy poskytovaly jednu vlastnost pro uchování hodnoty primárního klíče. Další omezení nejsou specifikována, takže tyto třídy zpravidla obsahují i další manipulační metody a často také implementují rozhraní *Serializable* za účelem jejich serializace.

4.5 Vaadin

Vaadin je rámec, který je navržen pro jednoduché vytváření a správu webového uživatelského rozhraní. Vaadin poskytuje server-driven model, díky kterému vytváříme dynamický web stejně, jako by jsme vytvářeli AWT, Swing nebo SWT aplikace.

Vaadin se skládá ze serverové části a klientské části. Klientská část je spuštěna v prohlížeči jako JavaScriptový program, který vykresluje uživatelské rozhraní a zajišťuje komunikaci s serverem. Vykreslování UI v prohlížeči je postaveno na technologii Google Web Toolkit (GWT), kde kód aplikace je psán v Javě a následně překompilován do JavaScriptu. Nutno zmínit, že díky překompilování a spouštění v prohlížeči jako JavaScriptový program, není pro zobrazení stránky v prohlížeči potřeba žádných zásuvných modulů, jak se třeba děje v případě jiných technologií jako je Flash či Java appplety [12].

Na serveru běží aplikace jako Java Servlet session a probíhá zde zpracování logiky uživatelského rozhraní společně s business logikou. Tyto dvě části poté spolu komunikují pomocí technologie AJAX.

Vaadin poskytuje většinu základních komponent pro tvorbu GUI, na které jsme zvyklí třeba z knihovny Swing. Práce s těmito komponentami je rovněž obdobná, kde vzhled jednotlivých komponent můžeme upravovat pomocí CSS.

5 Vývoj software a agilní přístup, práce v týmu

Protože jsem se poprvé podílel na vývoji většího projektu a také poprvé pracoval na vývoji v týmu, rozhodl jsem se také něco napsat o metodice vývoje softwaru, konkrétně o metodice Scrum, která nejvíce odpovídá mým zkušenostem s vývojem My Wires, u kterého by klasický vodopádový model zřejmě neuspěl.

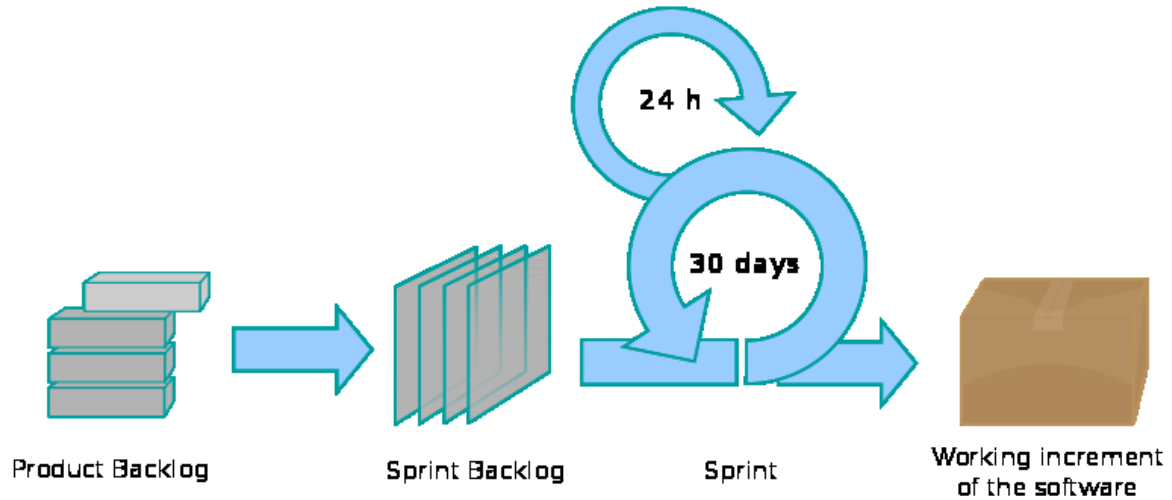
Agilní metodiky i agilní techniky historicky vycházejí z Agile Manifesto. Tento manifest vznikl na bázi dlouholetých zkušeností špičkových vývojářů, analytiků zdola nahoru. Tedy managementu navzdory. Přesto se postupy z něj vycházející osvědčily. [8]

5.1 Scrum

Scrum je relativně nová metoda agilního programování, která menším týmům dovoluje v krátkých iteracích vyvíjet produkty s dobře odhadnutelnou cenou a podle zákazníkových požadavků. Typický model metody Scrum předpokládá menší tým developerů, v rozmezí 4 – 15 členů. Jeden člen týmu je tzv. Scrum Master a pozicí se blíží běžnému projekt manažerovi, nicméně toto označení se pro něj nepoužívá. Další významnou osobou je Product Owner, který zastupuje stakeholdery, tedy zákazníka, který si daný produkt objednal. Posledním členem Scrum týmu je samotný tým "řadových" vývojářů - důraz na týmovou spolupráci je tu tedy evidentní, nicméně každý člen má velkou volnost a vystupuje vždy sám za sebe. [9]

Veškeré požadavky projektu jsou shromažďovány na tzv. Product backlog. To je jedna z hlavních výhod Scrumu - všechny požadavky jsou na jednom místě a nemusí se organizovat do hierarchie, jako se vesměs děje u jiných metod. V našem případě pro Product backlog sloužil systém Atlassian Jira, který Scrum podporuje. Dalším podobným instrumentem je tzv. Sprint backlog, který také obsahuje požadavky, ale v menším počtu a patřící do jednoho konkrétního Sprintu.

Scrum nabízí i jednoduchý prostředek, jakým lze odhadovat množství práce do ukončení projektu, tzv. Scrum Burn down. Jedná se o aproximaci množství zbývajících požadavků, které je přesnější s blížícím se koncem projektu, nicméně již od začátku nám může odhalit nereálnost aktuálního termínu dokončení a můžeme tím pádem provést potřebná opatření včas.



Obrázek 2 – Scrum [10]

5.1.1 Iterace

Práce v modelu Scrum je rozdělena na iterace, nazvané Sprinty, které trvají zpravidla mezi 1 a 4 týdny. Začátek každého Sprintu začíná výběrem požadavků, které se budou v dané iteraci plnit, jejich analýzy, implementace a předvedení zákazníkovi. Trvání naších iterací bylo dost proměnlivé, bylo období, kde byl čas jednotlivých iterací zkrácen na jeden týden. Vzhledem k tomu, že jsme byli přítomni jen 2 dny v týdnu, nebyla délka iterace 1 týden zrovna ideální, protože jsme nestačili dostatečně připravit scénáře na prezentaci výsledků dané iterace zákazníkovi.

5.1.2 Stand-up meeting

Po vzoru metodiky Scrum, bylo na každý den ve stejný čas naplánováno pravidelné krátké setkání týmu, kde každý člen musel v krátkosti zodpovědět tři otázky: co dělal včera, co bude dělat dnes a zda-li nastaly nějaké nenadálé problémy, které by mohly narušit plány dané iterace. Mezi zúčastněnými by měl být i Scrum Master.

5.1.3 Prezentace výsledků iterace a následná specifikace požadavků

Každá iterace je završena prezentací výsledků zákazníkovi, kterému jsou představeny změny a nová funkcionalita. Zákazník si výsledky prohlédne a dále určí zda-li mu dané řešení vyhovuje, nebo si přeje jej upravit. Po zhodnocení výsledků, zákazník dále specifikuje své požadavky, které by měly být implementovány v následující iteraci.

Každé prezentaci výsledků předchází příprava scénářů, podle kterých by se měla prezentace odvíjet. Příprava scénářů nám dělala problémy dlouhou dobu, ale nakonec jsme si zvykli dobře se připravit. U nás se projevila negativní stránka přípravy scénářů a to tou formou, že zbylo méně času na samotnou implementaci, protože dva pracovní dny v týdnu byly málo.

Rád bych ještě zmínil jeden docela užitečný zvyk, tzv. „**Code freeze“**, kdy se např. dva dny před prezentací pozastaví implementace nové funkcionality a začne se testovat stávající implementace, popřípadě se opravují nalezené chyby. Tento zvyk vychází ze zkušeností s vývojem softwaru, kde se na poslední chvíli tým snaží doimplementovat co nejvíc funkcionality, bohužel ale na úkor funkčnosti systému jako celku. Tento zvyk zde zmiňuji právě díky osobní zkušenosti, kde se nám několikrát stalo, že prezentace výsledků dané iterace skončila dříve než začala. Díky *NullPointerException* již při přihlašování do systému. Této zkušenosti před zákazníkem by se tedy měl každý tým vyvarovat a předcházet ji právě zmiňovaným pozastavením implementace a testováním.

5.2 Práce v týmu a SVN

Na vývoji se nás podílelo nějakou dobu i dvanáct členů. Tato skutečnost zahrnuje i některé úskalí, kterým tým musí čelit. Jako největší problém je synchronizace a rozdělení práce. Tento problém jsem vnímal hlavně z počátku, kdy se členové nově sestaveného týmu teprve poznávali a komunikace mezi jednotlivými členy tedy vázla. Navíc každý člen týmu měl jiné zkušenosti. Tento problém řešil projektový manažer, který pevně rozdělil úkoly a začalo se specifikovat, analyzovat a implementovat. Také byly naplánované tzv. Knowledge sharing sessions, kde zkušenější členové týmu předávali znalosti ostatním.

Pro správu verzí a sdílení zdrojových kódů v rámci týmu byl použit nástroj Subversion (SVN), který je následovník CVS. Tento nástroj se skládá ze serverové části, kde je centrální úložiště verzovaných souborů, a části klientské, která slouží pro práci s verzemi v souborovém systému. Díky tomu je umožněno vývojářům provádět změny v kódu na kopii projektu, která je uložena na jeho počítači. V případě otestovaných změn je pak vývojář může tzv. commitnout (odeslat) na server, kde se již změny projeví ostatním členům týmu pracujících na stejném projektu.

6 Zvolený postup řešení zadaných úkolů

Hlavním cílem bylo navrhnout a vytvořit základ informačního systému pro správu ethernetových zásuvek RJ-45. Cílovou skupinou systému by měly být společnosti se středně velkou síťovou infrastrukturou, rozdělenou do logických celků, ve které dochází k větší migraci uživatelů právě mezi těmito celky. Vzhledem k velké režii spojené s migrací uživatelů vznikají i bezpečnostní rizika, kde uživatel může nedopatřením získat přístup do privátní části sítě, do které by přístup mít neměl. Systém by měl pokrývat všechny možné scénáře plynoucí z těchto přesunů a poskytovat ucelený obraz aktuálního stavu síťových zdrojů. Způsob návrhu a řešení vycházel ze zkušeností s procesem migrace ve společnosti Tieto, která sídlí v několika geograficky vzdálených budovách, majíce spoustu různých oddělení a v každém oddělí spoustu různých týmů. Zvláště v oddělení JOS, kde sídlí Java developeři, docházelo k častým přesunům jednotlivců nebo i týmů do jiných kanceláří. Systém byl navrhován i s ohledem na požadované technologie.

6.1 Funkční specifikace systému

6.1.1 Role

Uživatelé vstupující do systému mohou mít různá práva. Z tohoto důvodu byly v systému definovány celkem čtyři základní role: člen týmu, tým leader, síťový specialista a administrátor. Vzhledem ke způsobu řešení autorizace, je počet rolí omezen pouze mohutnosti množiny definovaných práv. Nové role může definovat administrátor. V systému platí pravidlo, že uživatel sice může vystupovat vůči systému ve více rolích současně, ale v týmu může mít právě jen jednu roli.

6.1.1.1 Člen týmu

Je definován jako základní role v systému, která může být na základě povolení Team leadera rozšířena o další práva. Uživatelé s touto rolí mohou provádět tyto aktivity:

- Měnit nastavení svého profilu
- Prohlížet své zásuvky (aktivní požadavky)
- Vytvořit požadavek pro zapojení zásuvky
- V případě povolení Team leaderem:
    - Získat právo na automatické schvalování
    - Editovat nastavení zásuvek
    - Odstranit zásuvku

6.1.1.2 Team Leader

Vedoucí týmu je role, která je zodpovědná za správu týmu a dědí všechny aktivity definované pro člena týmu. Platí pravidlo, že v jednom týmu může být pouze jeden Team Leader. Aktivity definované pro tuto roli jsou následující:

- Přidávat členy do svého týmu
- Odebírat členy ze svého týmu
- Potvrzovat požadavky členů svého týmu
- Zamítat požadavky svého týmu
- Přidat nebo odebrat členovi svého týmu právo pro odstranění či změnu stavu svých zásuvek
- Přidat nebo odebrat členovi svého týmu právo pro automatické schvalování

6.1.1.3 Síťový specialista

Tato role je tzv. stand alone rolí v systému, která nedědí žádné aktivity ostatních rolí. Síťový specialista provádí fyzické zapojení zásuvek. Systém by tedy měl využívat pro sledování stavu síťových zdrojů. Aktivity uživatele s touto rolí jsou následující:

- Zobrazit všechny zásuvky v systému
- Vyhledávat v zásuvkách podle následujících kritérií
  - Budova
  - Číslo zásuvky
  - Login uživatele
  - Tým
  - Typ sítě

6.1.1.4 Administrátor

Administrátor je role pro správu systému, která může provádět tyto aktivity:

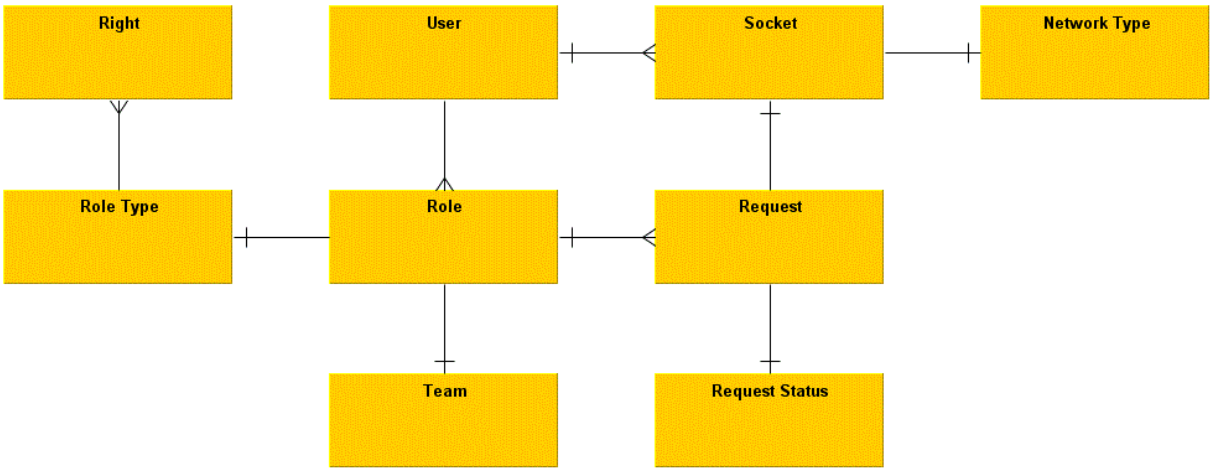
- Zobrazit seznam uživatelů aplikace
- Zobrazit seznam typů sítě
- Přidat uživatele do systému
- Přidat tým do systému
- Editovat profil uživatele
- Přidat uživatele do týmu

6.1.2 Use Case digramy

Vzhledem k velikosti diagramů případu užití, jsou diagramy umístěny v příloze.

6.1.3 ER diagram

Z analýzy požadavků vyplynul návrh datové základny, která je znázorněna na obrázku č.3 ER diagramem. Jednotlivý popis typů entit je rozebrán v kapitole 6.4 .



Obrázek 3 – ERD

6.2 Nefunkční specifikace systému

6.2.1 Specifikace technologií

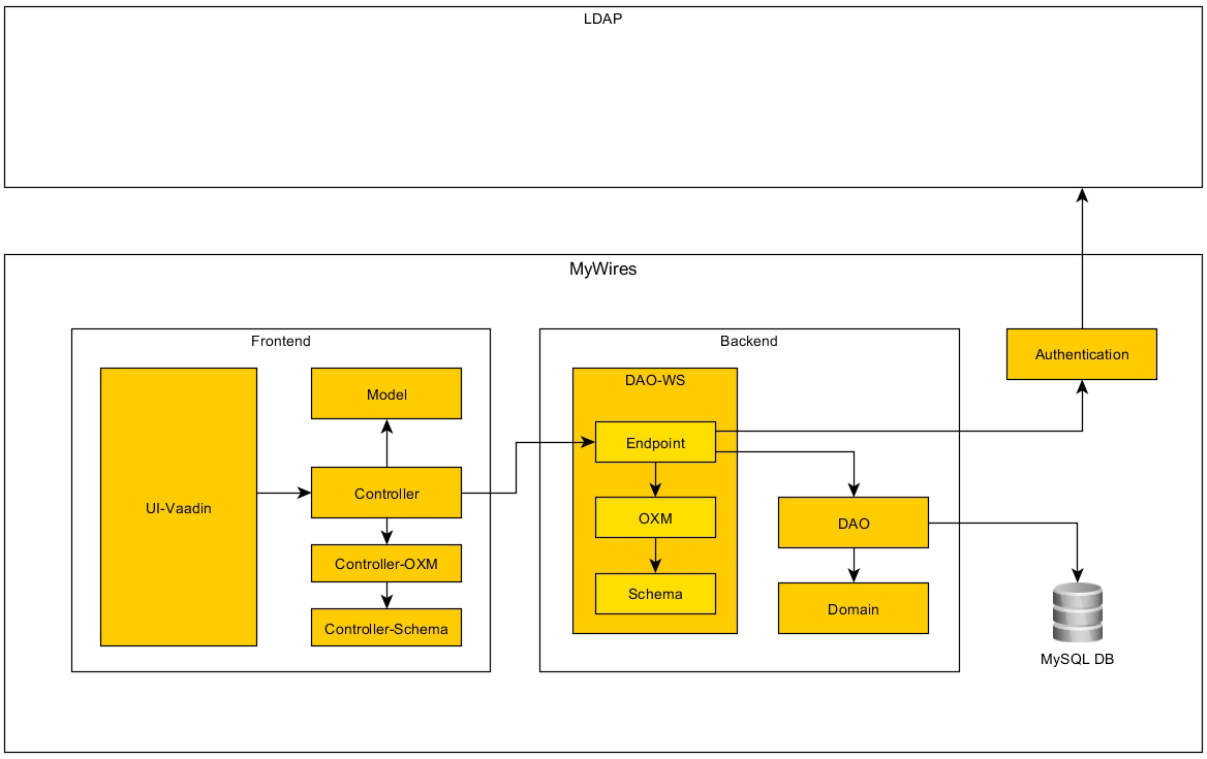
Systém má být implementován v jazyce Java a rozdělen na dva logické moduly, které spolu mají komunikovat za pomocí webových služeb. Pro vytvoření uživatelského rozhranní má být použit framework Vaadin, pro ORM framework Hibernate, který by měl zajistit nezávislost na databázi. Samotná cílová databáze není specifikována, nicméně funkcionalita systém by měla být otestována na databázovém enginu MySQL. Pro sestavování projektu má být použit Apache Maven a vlastnosti rámce Spring by měly být využívány v každém modulu.

6.2.2 Pokrytí testy

Dalším požadavkem je pokrytí jednotlivých modulů jednotkovými testy za pomocí JUnit. Cílové pokrytí by mělo být minimálně 80%.

6.3 Architektura systému

Celý systém je tedy podle požadavků rozdělen na dva logické moduly – na tzv. Backend modul a Frontend modul. Navzájem tyto dva moduly komunikují za pomocí webových služeb, skrze SOAP protokol. Díky oddělení těchto dvou modulů a vzhledem ke způsobu komunikace, můžeme mít frontendový modul nasazený na serveru A, backendový modul na serveru B a databázi na serveru C, kde servery A, B, C jsou na sobě geograficky nezávislé za předpokladu, že mezi nimi existuje TCP/IP spojení. Další výhodou této architektury je možnost mít libovolné grafické uživatelské rozhranní na libovolné platformě, která podporuje standard webových služeb. K tomu, abychom byli schopni toto libovolné UI napsat, nám stačí jen adresy webových služeb a kontrakt definující strukturu zpráv. Poté nám již nic nebrání ve vytvoření libovolného UI postaveného nad backend modulem.



Obrázek 4 - Architektura My Wires

6.4 Struktura balíčků a jejich popis

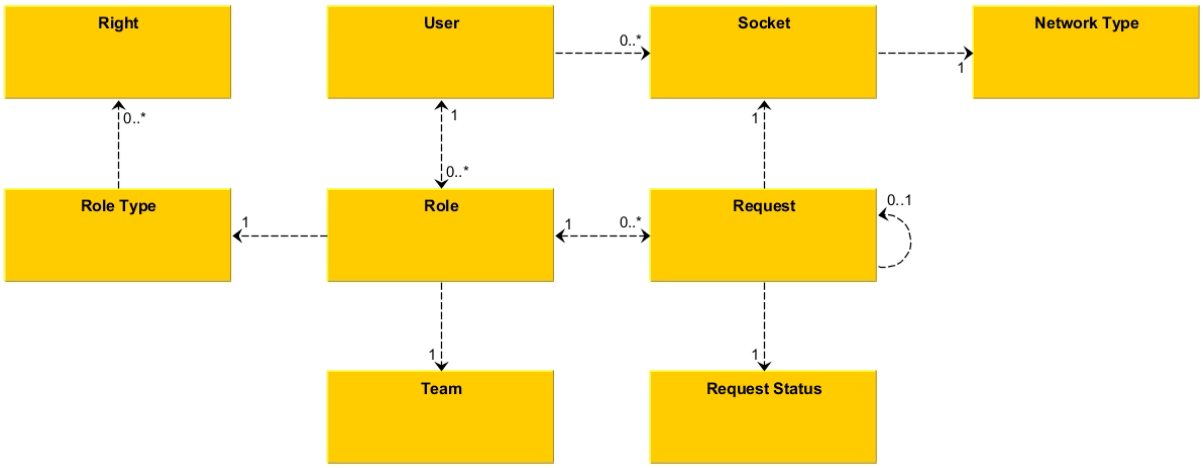
V této části bych rád trochu detailněji popsal obsah a význam jednotlivých balíčků, z kterých se systém My Wires skládá.

6.4.1 Backend modul

Jak již bylo řečeno, systém je rozdělen na dva logické moduly. Modul Backend obsahuje následující balíčky (submoduly): DAO-WS, DAO a Domain.

6.4.1.1 Domain

Doménový modul obsahuje doménové třídy aplikace, které slouží jako transfer objekty informací načtených z databáze či ukládaných do databáze. Jak můžeme vidět z třídního diagramu na obrázku č.5, v aplikaci jsme použili celkem 9 business tříd, což není moc. Vzhledem k malé doméně, jsme mohli daleko lépe navrhnout vztahy mezi jednotlivými typy entit, protože jak můžeme z třídního diagramu vypozorovat, obsahuje cyklus. Samotnému databázovému enginu cyklus nevadí, horší situace nastává v případě použití různý rámců jako je Hibernate pro ORM nebo třeba JAXB pro mapování objektů na XML a naopak, kde bez explicitního přerušení tohoto cyklu dochází k zacyklení.



Obrázek 5 - Třídní diagram

Ve třídě *Right*, která je výčtovým typem, jsou uložena všechna práva, vůči kterým se přihlášený uživatel autorizuje k přístupu k webovým zdrojům. Pro definování jednotlivých rolí v systému je určena třída *RoleType*. Tato třída agreguje určitou množinu práv a tím definuje jednotlivé role v systému.

Třída *Socket* reprezentuje zásuvku a uchovává informace jako je číslo místnosti, číslo zásuvky a zapouzdřuje informace o typu sítě. Typ sítě je reprezentován výčtovým typem *NetworkType*.

Ve třídě *User* jsou uchovány všechny potřebné informace o uživateli, jako je jeho login, jméno a příjmení. Dále jsou zde uloženy dodatečné informace o preferovaném barevném tématu či záložky a také, zda-li si přeje zobrazovat historii svých požadavků. Informace o hesle k přístupu do systému tato třída neobsahuje, protože se uživatel autentizuje vůči LDAP.

Stěžejní business třídy jsou *Role* a *Request*. Třída *Role* zapouzdřuje informace o uživateli, v jaké vystupuje roli v jakém týmu a načítá všechny požadavky daného uživatele.

Třída *Request* zapouzdřuje informace o zásuvce, datu vytvoření a stavu požadavku. Stav požadavku je reprezentován výčtovým typem *RequestStatus* a veškeré možné stavy požadavku jsou následující: aktivní, archivovaný, editovaný, zrušený, čekající na schválení, zamítnutý, pouze ke čtení a čekající na odstranění. Logiku řešení různých stavů požadavku jsem neřešil já, proto zde životní

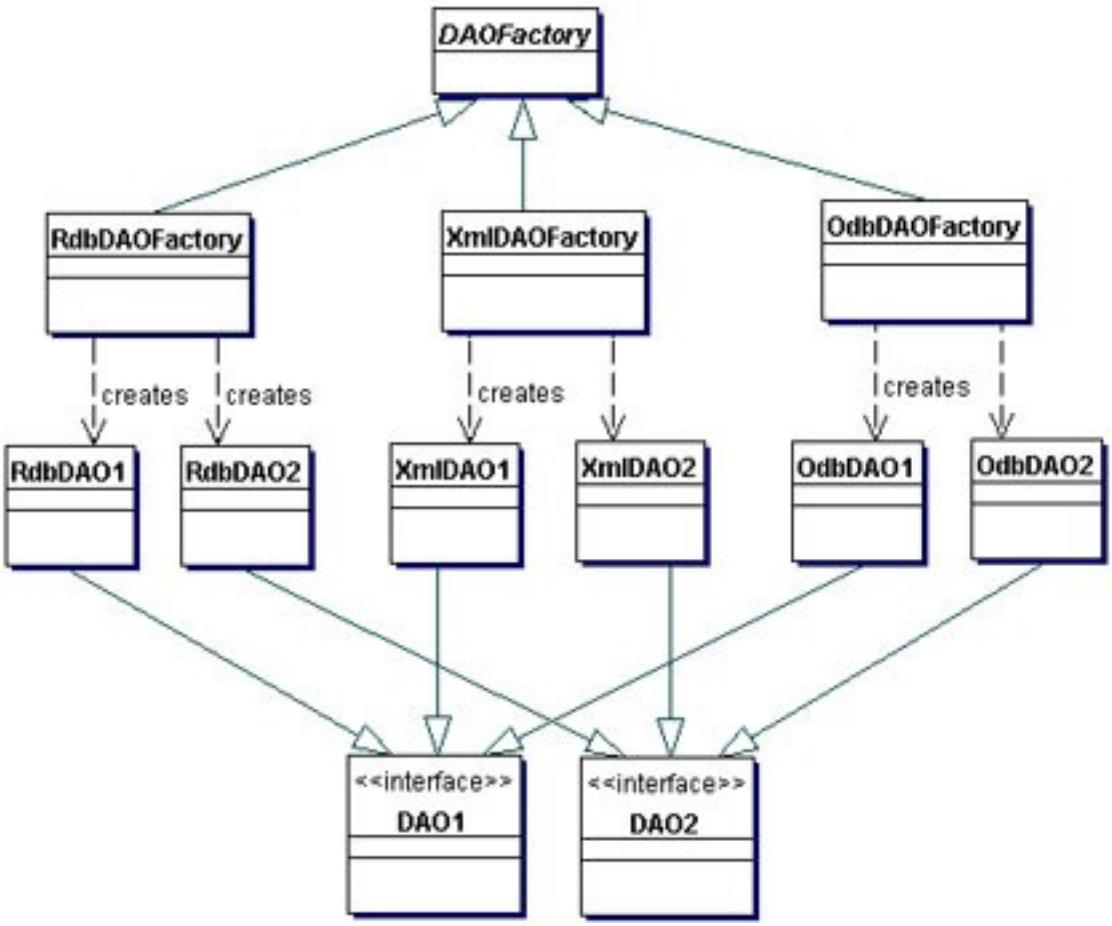
cyklus požadavku popisovat nebudu. Jen bych rád objasnil proč má třída Request odkaz sama na sebe, jak můžeme vidět v třídním diagramu na obrázku č. 5 . Je to z důvodu řízení souběhu, protože existuje reálná možnost, kde Tým leader může požadavek schválit respektive odmítnout v době, kdy Člen týmu požadavek zrovna edituje. Tím pádem by mohlo dojít k nekonzistenci dat. Tento problém je řešen tím způsobem, že při editaci je požadavek přepnut do editován a původní informace přejde do stavu pouze pro čtení aby se zabránilo ke ztrátě této informace v případě zamítnutí. Logika v systému zabraňuje potvrzení respektive zamítnutí požadavku se stavem editován a v případě pokusu je daný Team leader o této skutečnosti informován.

Tento modul je testy pokryt na 100%. Testují se zde *set* a *get* metody, jiné metody v doménových třídách nejsou.

6.4.1.2 DAO

V tomto modulu jsou prováděny všechny operace s databází. Pro ORM je zde použit rámec Hibernate, jehož konfigurace je uložena v *ressource* adresáři. V tomto adresáři je uložena jak konfigurace *SessionFactory*, tak XML konfigurace mapování perzistovaných objektů. V těchto konfiguračních souborech definujeme i způsob inicializace kolekce zapouzdřených objektů – Lazy nebo Eager. V XML konfiguraci perzistovaných objektů jsou uloženi i složitější HQL dotazy. Právě díky použití HQL dotazů, která jsou vždy validní, můžeme i nadále počítat s možnou změnou databáze. Pro účely testování jsme použili in-memory HSQLDB databázi, nicméně drobnou změnou konfiguračního souboru, lze použít libovolnou databázi..

Pro řízení přístupu k datovému zdroji byl využit návrhový vzor DAO, díky kterému můžeme systém bez změn ve vyšších vrstvách doplnit jiným zdrojem dat, třeba XML databází.



Obrázek 6 - Návrhový vzor DAO - Data Access Object [11]

Důležitá je abstraktní třída *DAOFactory*, která implementuje návrhový vzor Factory. Od *DAOFactory* dědí konkrétní tovární třídy pro perzistenci, ke který je přistupováno pomocí společného rozhraní. Tím jsme perfektně oddělení od konkrétní implementace. Této technice se říká programování proti rozhraní.

Pokrytí testy je splněno na 100%. Jak jednotkovými testy, tak integračními testy.

6.4.1.3 DAO-WS

V tomto modulu jsou další tři submoduly: Endpoint, Schema a OXM. Moduly Schema a OXM definují kontrakt webových služeb.

Důležitým modulem je Endpoint, vy kterém jsou definovány jednotlivé služby a právě tento modul se nasazuje na aplikační server. V tomto modulu se nacházejí ještě balíčky Security, Validators, Endpoints a Services. V balíčku Security je třída pro autentizaci uživatele vůči LDAP, pak je zde třída rovněž pro autentizaci, ale je určena pro testování, tudíž při její použití stačí, aby uživatel byl veden v databázi a je automaticky i bez hesla autentizován.

Validátory jsou zde pro validování příchozích a odchozích objektů z webových služeb.

Implementace funkcí a jejich logiky jsou v balíčku Services. Každá z těchto funkcí je z důvodu podpory transakcí označena anotací *@Transactional*. Tyto funkce jsou volány z balíčku Endpoint, ve kterých jsou definovány jednotlivé služby s využitím Spring WS API.

Stejně jako DAO modul, jsou zde testy skoro kompletní a pokrývají téměř 100%.

6.4.2 Frontend modul

Modul Frontend se skládá se submodulů: Model, Controller a UI-Vaadin. Struktura i logika tohoto modulu vychází z návrhového vzoru MVC.

6.4.2.1 Model

V tomto modulu je obsažena doménově specifická reprezentace informací, které jsou uložené v tzv. view objektech (VO). VO reprezentace doménového objektu neobsahuje některé informace, jako je třeba ID záznamu v databázi, když s ID v kontroléru nepracujeme a ani jej uživateli nezobrazujeme.

6.4.2.2 Controller

Tento modul odděluje UI od webových služeb. Jsou zde implementovány mappery, které mapují OXM objekty na VO objekty. Důležitým souborem zde je *mywires-controller-ws-context.xml*, ve kterém jsou adresy webových služeb, na které se zasílají požadavky.

V tomto modulu je v balíčcích Controller-OXM a Controller-Schema definován i kontrakt webových služeb, který je shodný s kontraktem v backend modulu.

Pokrytí testy je zde 60%, kde pokryté jsou pouze implementace mapperů.

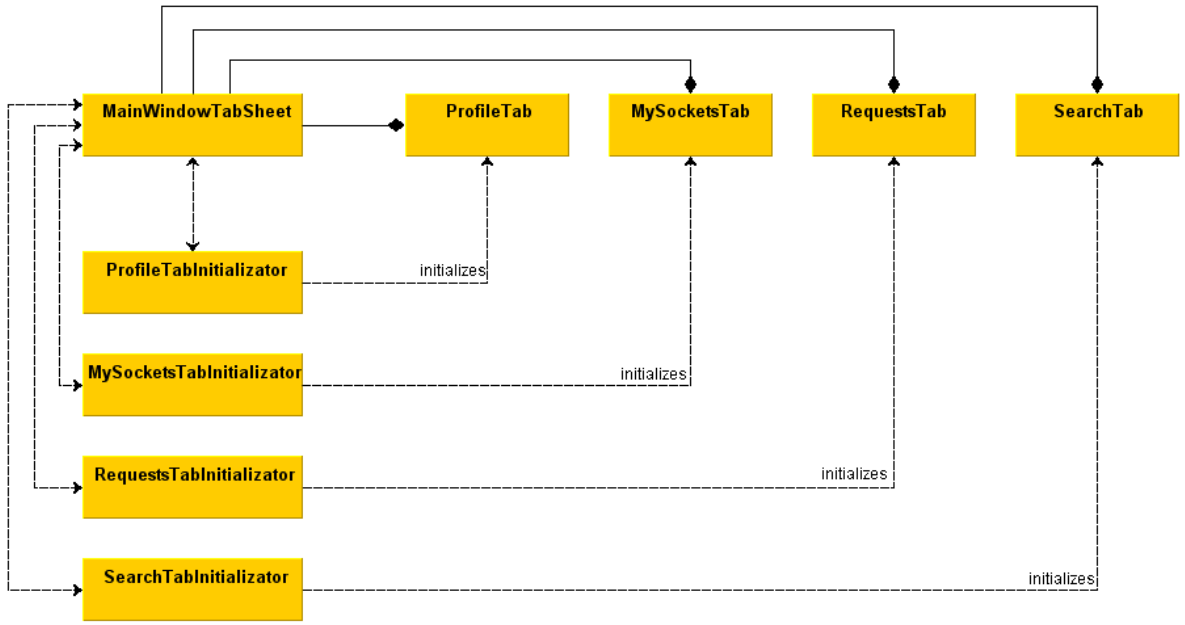
6.4.2.3 UI-Vaadin

Tento modul je zodpovědný za grafické rozhraní systému. Základem je třída *MyWiresApplication*, která dědí od třídy *Application* a díky tomu se chová jako servlet. Víceuživatelský přístup je řešen návrhovým vzorem ThreadLocale, díky kterému má každý uživatel vlastní instanci *MyWiresApplication*.

Výsledná podoba systému My Wires je v příloze na obrázcích č.10 – č.14 . Základem této podoby je třída *AbstractWindow*, která dědí od Vaadinovské třídy *Window*. Abstrakce byla zvolena z toho důvodu, aby všechny obrazovky měly stejný styl. V *AbstractWindow* je definována komponenta *Panel*, která obsahuje tři vertical layouty – header, body a footer. Dále obsahuje metody pro nastavení obsahu jednotlivých layoutů. Vizuální podoba těchto rozvržení je pak definována pomocí CSS stylů.

V případě prvního přístupu uživatele je do layoutu body načtena instance komponenty *LoginForm*, po přihlášení je načtena instance třídy *MainWindow*. Toto chování je definováno ve třídě *MyWiresApplication*. Jak *LoginForm*, tak *MainWindow* dědí od třídy *AbstractWindow*, právě kvůli zmiňovanému jednotnému stylu. *MainWindow* pak dále obsahuje komponentu *MainWindowTabSheet*, která dědí od Vaadinovské komponenty *TabSheet*. *MainWindowTabSheet* pak obsahuje jednotlivé záložky ve formě komponenty *Panel*.

Po přihlášení je provedena autorizace k přístupu k webovým zdrojům a to na základě práv uživatele, které byly načteny z databáze. Autorizace se provádí na úrovni jednotlivých záložek, tzv. tabů, kde pak dochází ještě k autorizaci na úrovni jednotlivých metod, které vykreslí UI buď s tlačítky nebo bez. Za tuto autorizaci jsou zodpovědné inicializátory komponent, které jsou na obrázku č. 7 .



Obrázek 7 – Inicializátory komponent

Nutno poznamenat, že autorizace je tzv. „**hard coded**“, jednoduchou podmínkou se testuje, zda-li uživatel má právo zobrazení dané komponenty, nebo zda-li má právo na zobrazení akčních tlačítek v dané komponentě. Ukázka kódu řešení takovéto autorizace je na následujícím výpisu, kde dochází k autorizaci uživatele na zobrazení tlačítek pro editaci a zrušení požadavků u tabulky požadavků:

```
public class MyRequestsTableInitializer extends AbstractInitializer {

    public void initializeComponent(Component parentComponent, UserVO userVO) {
        if (readUserRights(userVO).contains(RightEnumVO.REQUEST_OWN_CANCEL.getValue())
           || readUserRights(userVO).contains(RightEnumVO.REQUEST_OWN_EDIT.getValue())) {
            ((MySocketsTab) parentComponent).initTableMyRequestsWithActionButtons();
        }
        else if (readUserRights(userVO).contains(RightEnumVO.REQUEST_OWN_VIEW.getValue())) {
            ((MySocketsTab) parentComponent).initTableMyRequests();
        }
    }
}
```

Tento modul je také zodpovědný za lokalizace do českého a anglického jazyka. Lokalizace jsou v balíčku *resources*. Rozpoznání jazyka, který má být uživateli zobrazen, probíhá automaticky podle nastavení jazyka prohlížeče.

Uživatel si také v profilu může zvolit ze tří různých barevných témat.

V tomto modulu se bohužel testy nenacházejí.

6.5 Spoluautorství na projektu

Podílel jsem se na implementaci jednotkových a integračních testů DAO modulu, implementaci mapperů, které mapovaly OXM objekty na VO objety a naopak, implementaci dvou inicializátorů a refactoringu uživatelských práv. Dále jsem řešil chybu v systému, která se vyskytovala při editaci požadavků, kde bylo možno uložit editovaný požadavek na zásuvku, která již byla používána.

Největší podíl mám na výsledném rozvržení (layout) UI systému My Wires, které je rozvrženo jako klasická webová stránka na hlavičku, obsah a patičku, umístění loga společnosti Tieto, název systému a informace o přihlášeném uživateli. Výsledný systém je na obrázcích umístěných v příloze. Další mou prací je, že uživatel má možnost si zvolit v profilu, zda-li si přeje zobrazovat historii svých požadavků. Tento úkol představoval implementaci logiky ve všech modulech, od domény až po uživatelské rozhraní, včetně testů. Dále jsem vytvořil import SQL skript, který naplní databázi reálnými daty, kde je kladen důraz na pokrytí všech možností, které mohou v systému nastat. V poslední řadě bych rád uvedl, že posledních 7 dní jsem pracoval na záložce pro správu týmu, bohužel se mi nepodařilo implementaci dotáhnout do konce. Záložka pro editaci týmu je na straně Backendu hotová – Endpointy pro získání všech členů týmu na základě id týmu a získání všech uživatelů, kteří mohou být do týmu přidáni. Na straně Frontendu je hotová i záložka s tabulkou se všemi členy daného týmu. Nicméně chybí podpora pro plnění tabulky daty, to z důvodu cyklické závislosti Role > User > Role v implementaci mapperů v controller modulu, které provádějí mapování OXM objektů na VO objekty.

Obrázky č.3, č.4, č.5, č.7, č.8, č.9 uvedené v této práci, jsou převzaté z dokumentace systému My Wires.

7 Znalosti a dovednosti získané během studia a uplatněné v průběhu odborné praxe

Na pozici softwarového vývojáře v jazyce Java jsem samozřejmě využil znalosti nabyté ve velice povedené skladbě předmětu Programovací jazyky I, kde se probíraly pokročilejší techniky programování v jazyce Java včetně do hloubky probraného objektově orientovaného přístupu programování. Na pevné základy z PJ I se pak opíraly znalosti nabyté v předmětu Vývoj informačních systémů (VIS), kde se probíraly různé architektury informačních systémů, návrhové vzory, postupy návrhu a implementace vrstveného softwarového díla.

Velice užitečné se ukázaly také znalosti z předmětu Databázové a informační systémy (DAIS), ve kterém jsem pochopil problémy spojené s objektově-relačním mapováním (ORM), které dnes tvoří základ každé aplikace či systému pracujícím s databází. Na projektu jsme sice pro ORM požili rámec Hibernate, ale i ten je potřeba vhodně nakonfigurovat a použít. Také problematika řízení souběhu, transakce či samotné SQL probírané v DAIS se ukázaly jako cenné znalosti.

Dále pak nemohu vynechat znalosti nabyté v Softwarovém inženýrství (SWI), které mi pomohly ve čtení UML diagramů a následné implementace. Také při psaní dokumentace se znalost UML ukázala jako velice užitečná.

8 Znalosti a dovednosti scházející studentovi v průběhu odborné praxe

Systém jsme vyvíjeli pod platformou Java EE, pro vývoj sdílené business logiky byl použit framework Spring, pro přístup k legacy systémům Hibernate, front-endová část komunikuje s back-endovou částí skrze webové služby pomocí SOAP protokolu, na mapování objektu na XML a naopak je použita technologie JAXB, UI je napsané ve webovém frameworku Vaadin a celý projekt je sestavován pomocí Apache Maven.

Všechny výše uvedené technologie pro mě byly velkou neznámou. Je v celku zřejmé, že se jedná o velice úzkou specializaci a není tedy možné, abych se se všemi těmito technologiemi v průběhu studia setkal.

Jako svou velkou chybu považuji to, že jsem nenavštěvoval předměty Java technologie (JAT), kde se část specifikace Java EE probírala a dále Vývoj internetových aplikací (VIA), ve kterém bych přišel do styku s webovými službami, pochopil do hloubky HTTP protokol a naučil se základy JavaScriptu.

Dalším nedostatkem, na který jsem v průběhu odborné praxe narazil, bylo psaní UNITových a integračních testů, které se ukázaly jako nezbytné u rozsáhlejších projektů. Nejednou nám testy ušetřily námahu při hledání chyb. Myslím, že předmět věnovaný problematice testování softwaru by byl taky velice užitečný.

9 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Celý koncept odborné praxe ve firmě jako alternativu k bakalářské práci hodnotím velmi kladně. Již po absolvování Tieto IT Academy jsem si uvědomil, že teorie se od praxe podstatně liší a nic člověka nenaučí tolik, jako samotná praxe. Vzhledem k tomu, že jsem se poprvé zúčastnil vývoje většího softwarového projektu, byla pro mě odborná praxe velkým přínosem.

Mezi hlavní přínosy řadím zkušenosti s vývojem v týmu, iterativním vývojem softwaru, seznámení se v praxi s technologiemi použitými na projektu „My Wires“ a v neposlední řadě také povinnost dopracování zadané funkcionality až do konce, včetně otestování a předvedení zákazníkovi. Dalším neocenitelným přínosem byla možnost konzultovat řešení se zkušenějšími kolegy a to buď v týmu, nebo i klidně mimo něj.

Jak již bylo psáno dříve v této práci, cílem bylo vytvořit funkční a odladěný základ modulárního, dobře škálovatelného informačního systému pro evidenci stavu a správu zásuvek RJ-45. Celý systém se nám podařilo víceméně naimplementovat podle zadání a očekávání, nicméně čas na doimplementování poslední role Administrátora nezbyl. Implementována podpora v systému je tedy pro role: člen, člen týmu, tým leadera a síťového administrátora.

Celkově bych hodnotil absolvování individuální odborné praxe jako úspěšnou a přínosnou. Bylo mi ctí potkat spoustu zkušených lidí, kteří nešetřili cennými radami a neostýchali se podělit o své zkušenosti. Budoucím studentům naší fakulty mohu absolvování individuální odborné praxe ve společnosti Tieto Czech s.r.o. jen vřele doporučit.

10 Seznam obrázků

11 Literatura

[1] Tieto, *Informace o Tieto* [online], [cit 2012-04-16].
Dostupné z: < http://www.tieto.cz/o-nas >.

[2] Tieto, *Historie - Tieto - Czech Republic* [online], [cit 2012-04-16].
Dostupné z: < http://www.tieto.cz/o-nas/historie >.

[3] Apache Maven. *Wikipedia: the free encyclopedia* [online], St. Petersburg (Florida) : Wikipedia Foundation, [cit 2012-05-01].
Dostupné z: < http://cs.wikipedia.org/wiki/Apache_maven >

[4] Ferschmann, Petr. *Proč Maven* [online], [cit 2012-05-03].
Dostupné z: < http://blog.softeu.cz/proc-maven >.

[5] Matulík, Petr – Páral, Tomáš. *Moderní JEE™ technologie a nástroje* [online], ©2007, [cit 2012-04-29]. Dostupné z: < http://www.morosystems.cz/java/index.php >.

[6] Spring Framework. *Wikipedia: the free encyclopedia* [online], St. Petersburg (Florida) : Wikipedia Foundation, [cit 2012-05-02].
Dostupné z: < http://cs.wikipedia.org/wiki/Spring_Framework >

[7] Kosek, Jiří. *Využití webových služeb a protokolu SOAP při komunikaci* [online], ©2002, [cit 2012-04-28].
Dostupné z: < http://www.kosek.cz/diplomka/html/websluzby.html >.

[8] Knesl, Jiří. *Agilní vývoj: Úvod - Zdroják* [online], ©2009, [cit 2012-04-28].
Dostupné z: < http://www.zdrojak.cz/clanky/agilni-vyvoj-uvod >.

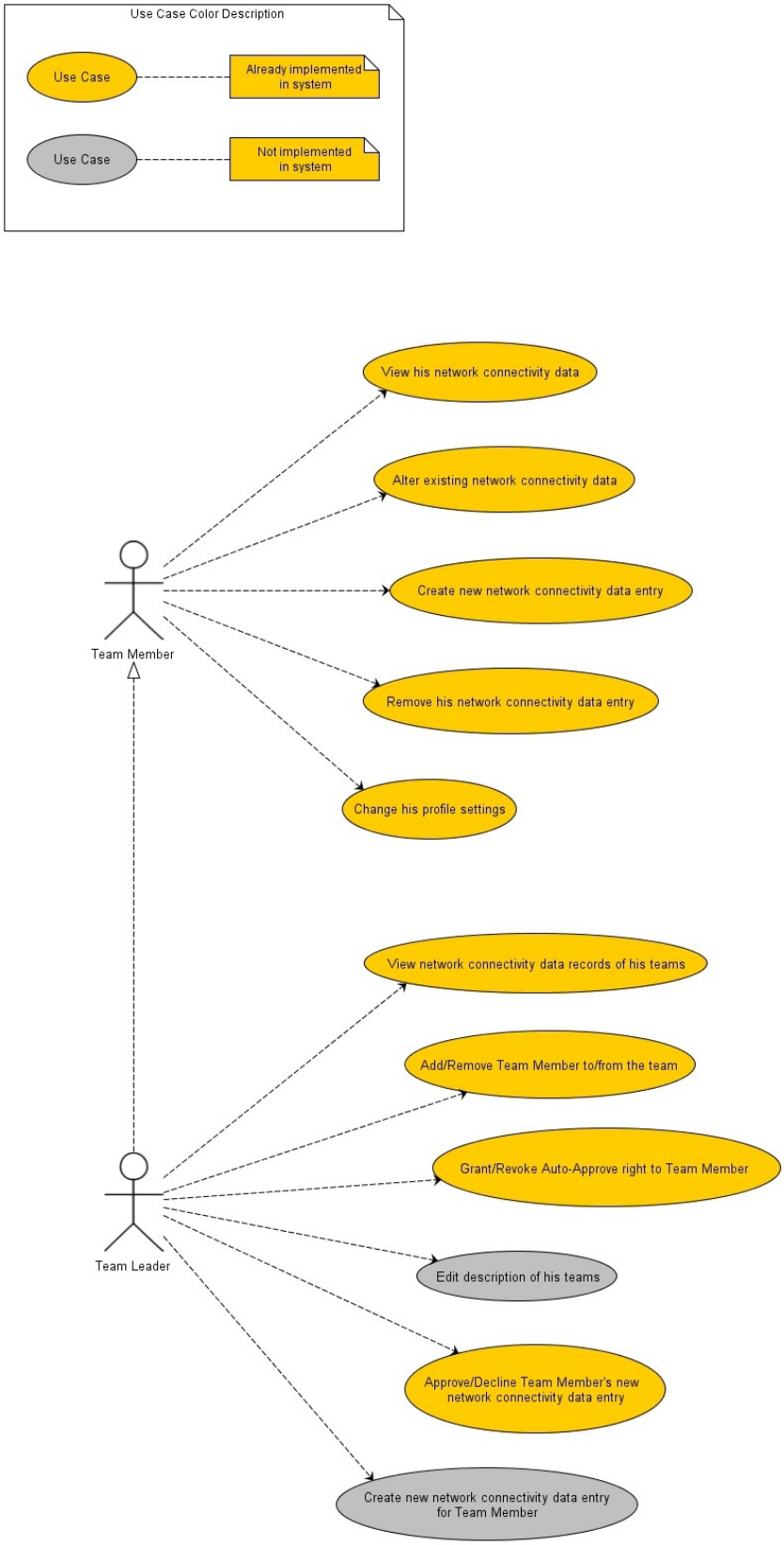
[9] Horák, Jan. *Scrum - metoda agilního programování* [online], ©2012, [cit 2012-04-28].
Dostupné z: < http://wild-web.eu/blog/scrum-metoda-agilni-programovani >.

[10] Scrum (development). *Wikipedia: the free encyclopedia* [online], St. Petersburg (Florida) : Wikipedia Foundation, [cit 2012-05-02].
Dostupné z: < http://en.wikipedia.org/wiki/Scrum_(development) >

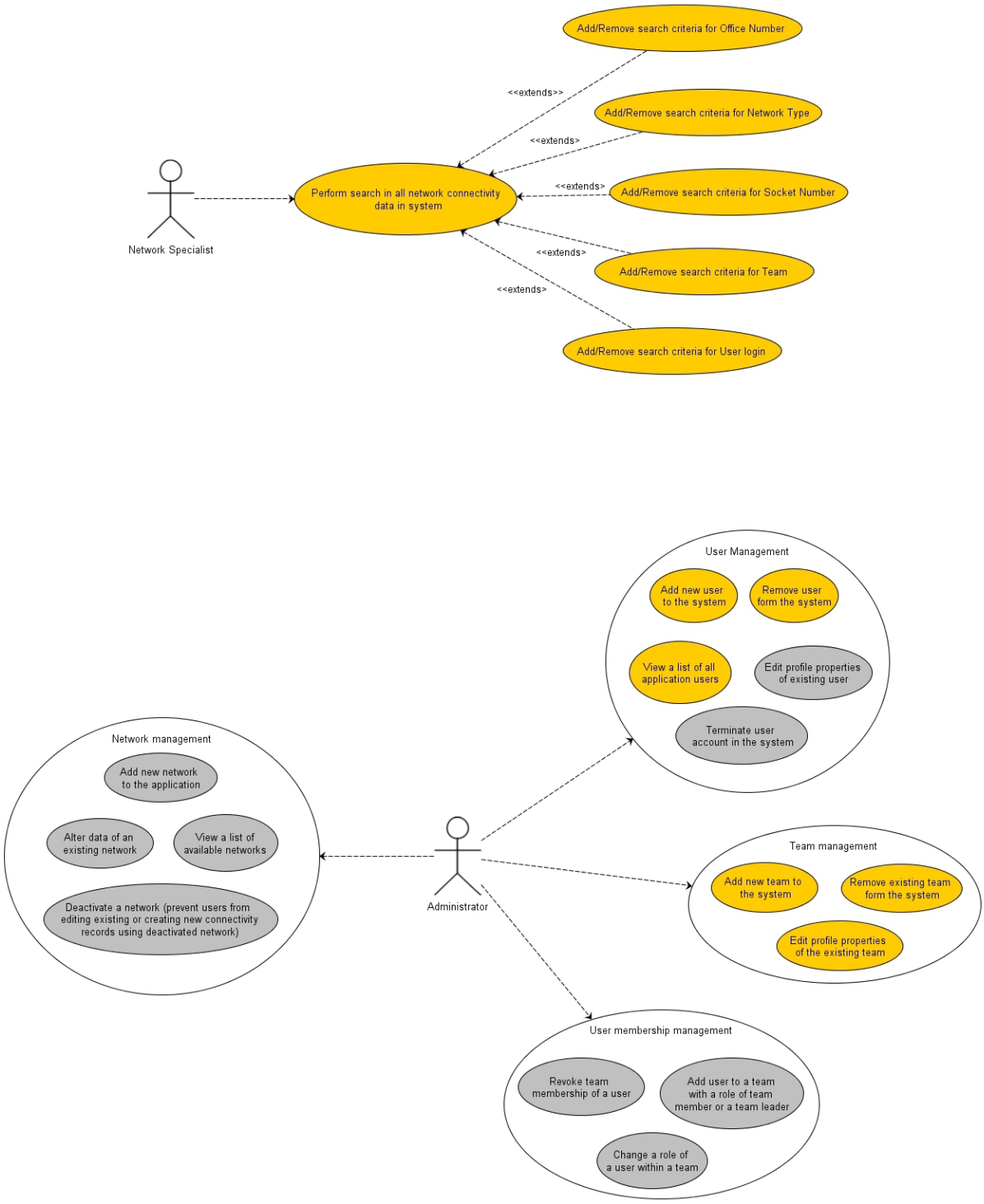
[11] Oracle. *Core J2EE Patterns - Data Access Object* [online], ©2010, [cit 2012-05-03].
Dostupné z:
< http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html >

[12] Vaadin Ltd. *Book of Vaadin* [online], ©2011, [cit 2012-05-03].
Dostupné z: < https://vaadin.com/book/-/page/intro.html >

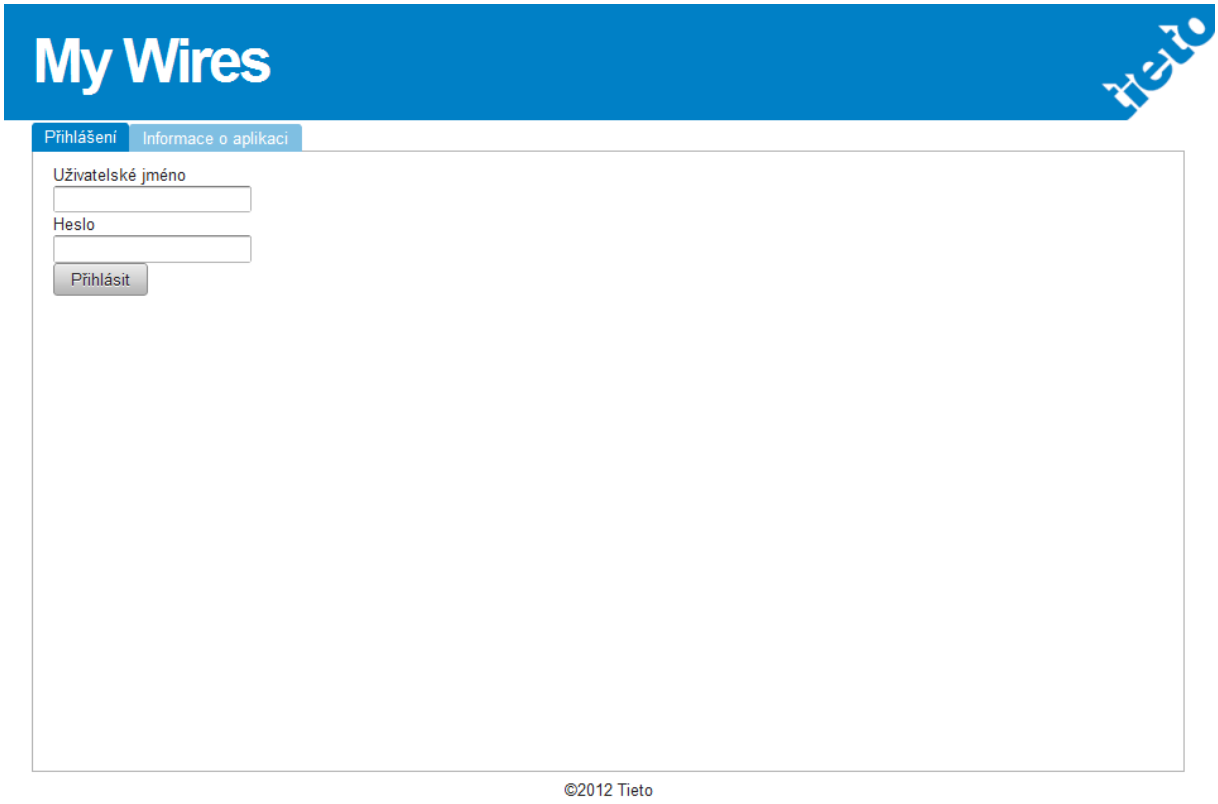
Přílohy



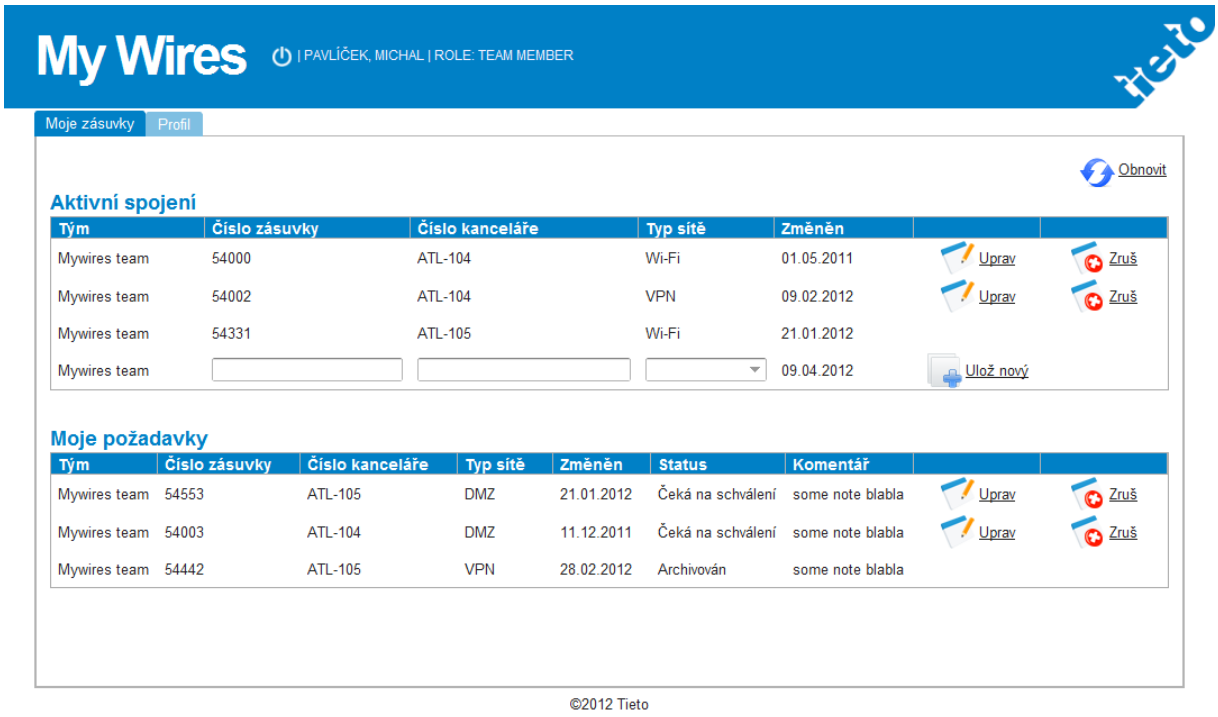
Obrázek 8 - Use Case diagram pro role Člen týmu a Tým Leadera



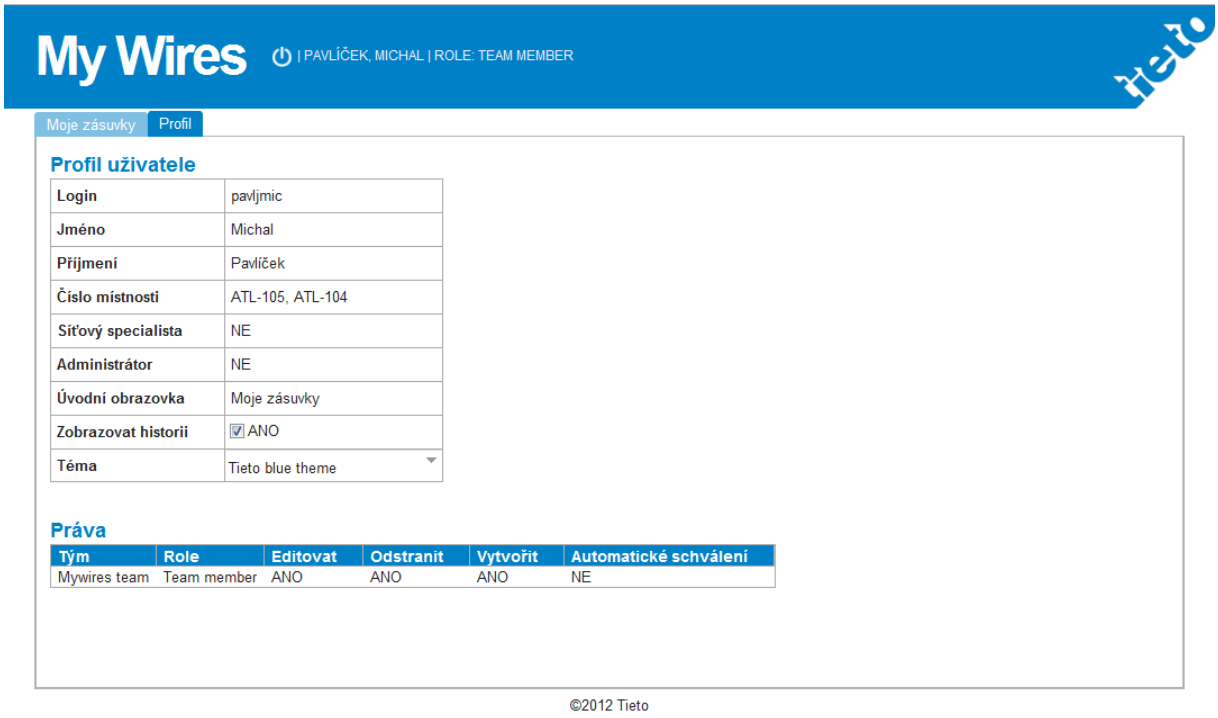
Obrázek 9 - Use Case diagram pro role Síťový specialista a Administrátor



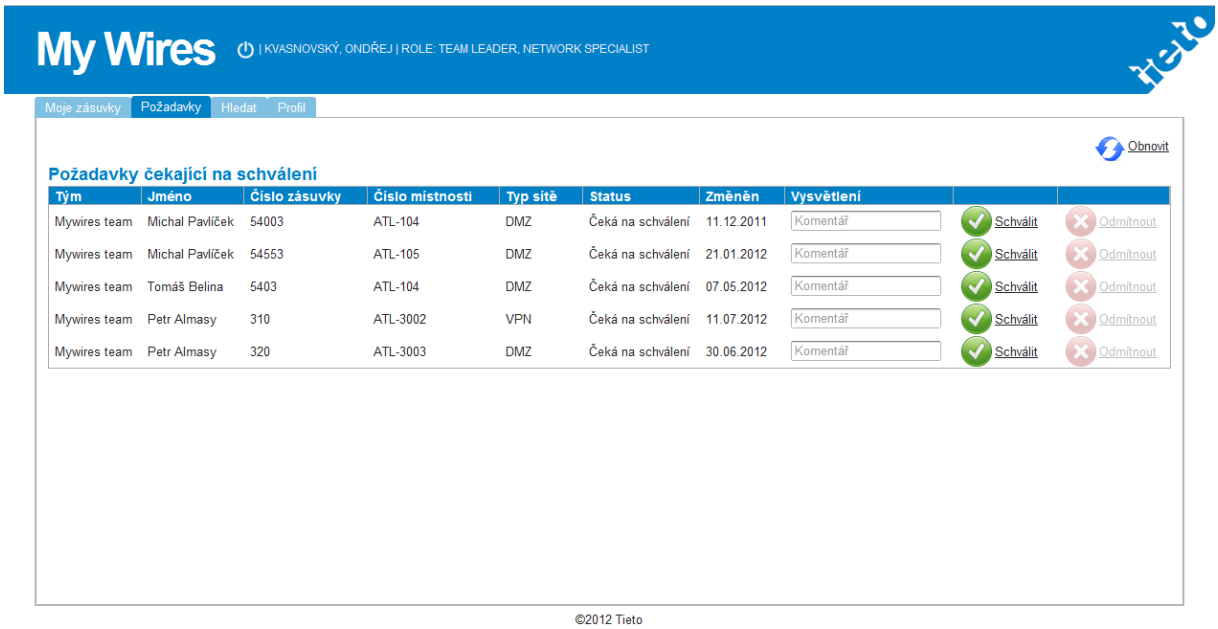
Obrázek 10 - Přihlašovací obrazovka



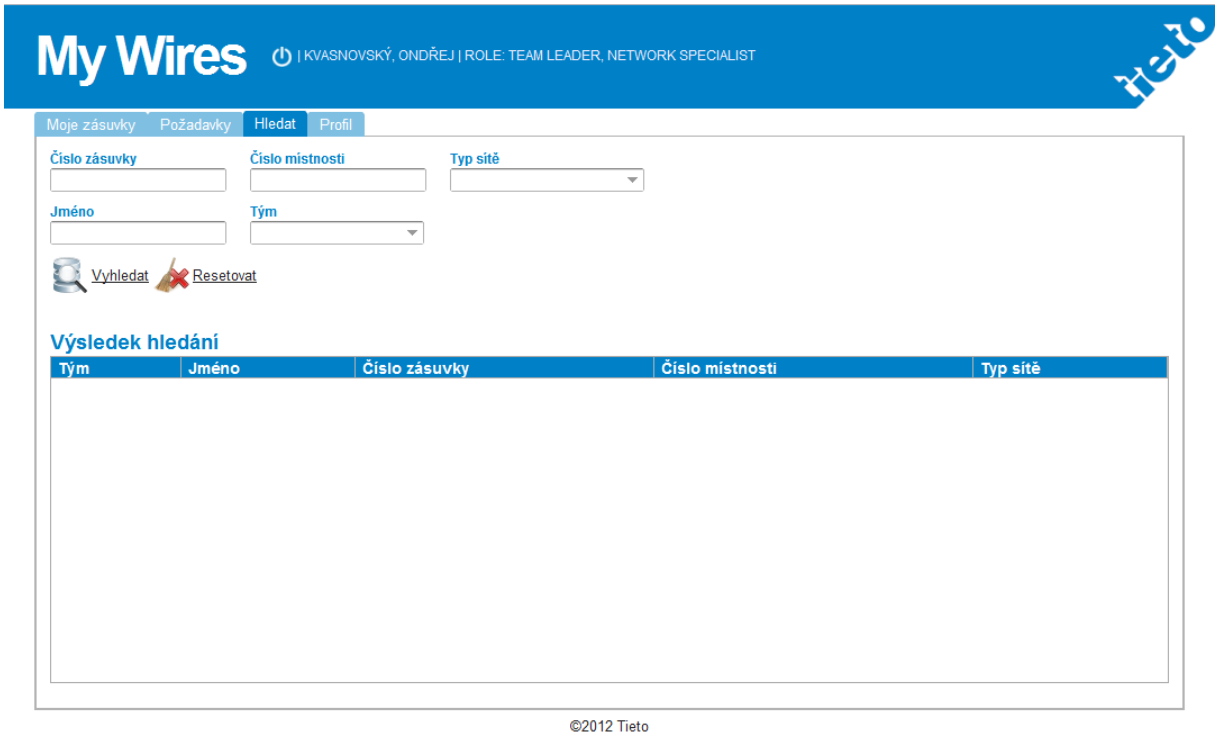
Obrázek 11 - Úvodní obrazovka Člena týmu



Obrázek 12 - Profil Člena týmu



Obrázek 13 - Zobrazení požadavků Tým Leaderem



Obrázek 14 - Vyhledávací formulář Síťového specialisty